Roadmap

C:
```c
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:
```java
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg = c.getMPG();
```

Assembly language:
```assembly
get_mpg:
    pushq %rbp
    movq %rsp, %rbp
    ...
    popq %rbp
    ret
```

Machine code:
```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

OS:
```
Windows & Mac
```

Computer system:
```
Memory & data
Integers & floats
```

Basics of Machine Programming and Architecture

- What is an ISA (Instruction Set Architecture)?
- A brief history of Intel processors and architectures
- C, assembly, machine code
- x86 basics: registers

Translation

Translation Impacts Performance

- The time required to execute a program depends on:
  - The program (as written in C, for instance)
  - The compiler: what set of assembler instructions it translates the C program into
  - The instruction set architecture (ISA): what set of instructions it makes available to the compiler
  - The hardware implementation: how much time it takes to execute an instruction

What makes programs run fast?

What should the HW/SW interface contain?
Instruction Set Architectures

- The ISA defines:
  - The system’s state (e.g. registers, memory, program counter)
  - The instructions the CPU can execute
  - The effect that each of these instructions will have on the system state

![CPU, Memory, Registers, PC diagram]

General ISA Design Decisions

- Instructions
  - What instructions are available? What do they do?
  - How are they encoded?

- Registers
  - How many registers are there?
  - How wide are they?

- Memory
  - How do you specify a memory location?

x86

- Processors that implement the x86 ISA completely dominate the server, desktop and laptop markets

- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on

- Complex instruction set computer (CISC)
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - (as opposed to Reduced Instruction Set Computers (RISC), which use simpler instructions)

Intel x86 Evolution: Milestones

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>29K</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>386</td>
<td>1985</td>
<td>275K</td>
<td>16-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium 4F</td>
<td>2005</td>
<td>230M</td>
<td>2800-3800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel x86 Processors

Machine Evolution
- 486 1989 1.9M
- Pentium 1993 3.1M
- Pentium/MMX 1997 4.5M
- PentiumPro 1995 6.5M
- Pentium III 1999 8.2M
- Pentium 4 2001 42M
- Core 2 Duo 2006 291M
- Core i7 2008 731M

Added Features
- Instructions to support multimedia operations
  - Parallel operations on 1, 2, and 4-byte data
- Instructions to enable more efficient conditional operations
- More cores!

More information

References for Intel processor specifications:
- Intel’s “automated relational knowledgebase”:
  - http://ark.intel.com/
- Wikipedia:

x86 Clones: Advanced Micro Devices (AMD)

Same ISA, different implementation

Historically
- AMD has followed just behind Intel
- A little bit slower, a lot cheaper

Then
- Recruited top circuit designers from Digital Equipment and other downward trending companies
- Built Opteron: tough competitor to Pentium 4
- Developed x86-64, their own extension of x86 to 64 bits

Intel’s Transition to 64-Bit

Intel attempted radical shift from IA32 to IA64 (2001)
- Totally different architecture (Itanium) and ISA than x86
- Executes IA32 code only as legacy
- Performance disappointing

AMD stepped in with evolutionary solution (2003)
- x86-64 (also called “AMD64”)

Intel felt obligated to focus on IA64
- Hard to admit mistake or that AMD is better

Intel announces “EM64T” extension to IA32 (2004)
- Extended Memory 64-bit Technology
- Almost identical to AMD64!

Today: all but low-end x86 processors support x86-64
- But, lots of code out there is still just IA32
Our Coverage in 351

- **IA32**
  - The traditional 32-bit x86 ISA

- **x86-64**
  - The new 64-bit x86 ISA – all lab assignments use x86-64!

Definitions

- **Architecture**: (also instruction set architecture or ISA)
  - The parts of a processor design that one needs to understand to write assembly code
    - “What is directly visible to software”

- **Microarchitecture**: Implementation of the architecture
  - CSE 352

- Is cache size “architecture”?
- How about CPU frequency?
- And number of registers?

Assembly Programmer’s View

- **Programmer-Visible State**
  - **PC**: Program counter
    - Address of next instruction
    - Called “EIP” (IA32) or “RIP” (x86-64)
  - **Register file**
    - Heavily used program data
  - **Condition codes**
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code, user data, (some) OS data
  - Includes stack used to support procedures (we’ll come back to that)

Turning C into Object Code

- **Code in files** `p1.c` `p2.c`
- **Compile with command**: `gcc -O1 p1.c p2.c -o p`
  - Use basic optimizations (–O1)
  - Put resulting machine code in file `p`
  - **Text**: C program
    - Compiler (`gcc -S`)
  - **Text**: Asm program
    - Assembler (`gcc` or `as`)
  - **Binary**: Object program
    - Linker (`gcc` or `ld`)
  - **Binary**: Executable program
    - Static libraries (`.a`)

```
## Compiling Into Assembly

### C Code

```c
int sum(int x, int y) {
    int t = x+y;
    return t;
}
```

### Generated IA32 Assembly

```assembly
sum:
pushl %ebp
movl %esp,%ebp
movl 12(%ebp),%eax
addl 8(%ebp),%eax
movl %ebp,%esp
popl %ebp
ret
```

## Machine Instruction Example

### C Code

```
int t = x+y;
```

### Assembly

```
addl 8(%ebp),%eax
```

- Similar to expression: \( x += y \)
- More precisely:
  ```assembly
  int eax;
  int *ebp;
  eax += ebp[2]
  ```

## Object Code

- **Code for sum**:  
  - Total of 13 bytes
  - Each instruction 1, 2, or 3 bytes
  - Starts at address 0x040140
  - Not at all obvious where each instruction starts and ends

## Disassembling Object Code

### Disassembled

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00401040</td>
<td>05</td>
<td>push</td>
<td>%ebp</td>
</tr>
<tr>
<td>0x1</td>
<td>89  e5</td>
<td>mov</td>
<td>%esp,%ebp</td>
</tr>
<tr>
<td>0x3</td>
<td>8b  45  0c</td>
<td>mov</td>
<td>0xc(%ebp),%eax</td>
</tr>
<tr>
<td>0x6</td>
<td>03  45  08</td>
<td>add</td>
<td>0x8(%ebp),%eax</td>
</tr>
<tr>
<td>0x9</td>
<td>89  ec</td>
<td>mov</td>
<td>%ebp,%esp</td>
</tr>
<tr>
<td>0x10</td>
<td>5d</td>
<td>pop</td>
<td>%ebp</td>
</tr>
<tr>
<td>0x11</td>
<td>c3</td>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

### Assembler

- Translates .s into .o
- Binary encoding of each instruction
- Nearly-complete image of executable code
- Missing links between code in different files

### Linker

- Resolves references between object files and (re)locates their data
- Combines with static run-time libraries
  - E.g., code for malloc, printf
- Some libraries are **dynamically linked**
- Linking occurs when program begins execution

### Disassembler

- Useful tool for examining object code (man 1 objdump)
- Analyzes bit pattern of series of instructions (delineates instructions)
- Produces near-exact rendition of assembly code
- Can be run on either p (complete executable) or p1.o/p2.o file
**Alternate Disassembly**

<table>
<thead>
<tr>
<th>Object</th>
<th>Disassembled</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x401040:</td>
<td>push %ebp</td>
</tr>
<tr>
<td>0x401041:</td>
<td>mov %esp,%ebp</td>
</tr>
<tr>
<td>0x401043:</td>
<td>mov 0xc(%ebp),%eax</td>
</tr>
<tr>
<td>0x401046:</td>
<td>add 0x8(%ebp),%eax</td>
</tr>
<tr>
<td>0x401049:</td>
<td>mov %ebp,%esp</td>
</tr>
<tr>
<td>0x40104b:</td>
<td>pop %ebp</td>
</tr>
<tr>
<td>0x40104c:</td>
<td>ret</td>
</tr>
</tbody>
</table>

**Within gdb debugger**
- `gdb` `p`  
  `disassembler sum`  
  (disassemble function)  
- `x/13b sum`  
  (examine the 13 bytes starting at `sum`)

**What Can be Disassembled?**

```
% objdump -d WINWORD.EXE
WINWORD.EXE:  file format pe-i386
No symbols in "WINWORD.EXE".
Disassembly of section .text:
30001000 <.text>:  push %ebp
30001001:  55 push %ebp
30001001:  8b ec mov %esp,%ebp
30001003:  6a ff push $0xffffffff
30001005:  68 90 10 00 30 push $0x30001090
3000100a:  68 91 dc 30 30 push $0x304cdc91
```

- Anything that can be interpreted as executable code
- Disassembler examines bytes and reconstructs assembly source